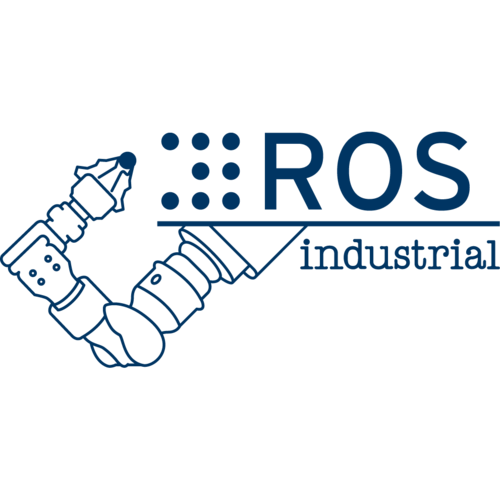
Notes of ROS



* Day 1

**1. Concept**

ROS is a flexible framework that is widely used in robotics. It is a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic plataforms.

For the creation of a robust robot system is someone very hard of applied, then from the robot’s perspective, problems that seem trivia to humans often vary wildly between instances of tasks and enviroments . Dealing with these variations is so hard that no single individual, laboratory, or institutions can hope to do on their own.

Ros was built from the ground up to encourage collaborative robotics software development .



*ROS.org*

ROS was originally developed in 2007 by the Stanford Artificial Intelligence Laboratory (SAIL) with the support of the Stanford AI Robot project. As of 2008, development continues primarily at Willow Garage, a robotics research institute, with more than 20 institutions collaborating within a federated development model.

A lot of research institutions have started to develop projects in ROS by adding hardware and sharing their code samples. Also, the companies have started to adapt their products to be used in ROS.

*Learning ROS for Robotics Programming*

* *Day 2*

Ros provide standard operating system

***2.* Why we prefer ROS for robotics?**

* **Capacidades de Alta gama:** ROS viene listo con capacidades para ser usado para la localización y mapeo simultaneo (SLAM) estos paquetes pueden ser usados para navegación autónoma amaestrada en robots móviles otro paquete disponible es MoveIT usado para planeación de movimientos de robots manipuladores. Todas estas capacidades pueden ser usadas directamente en el software de nuestro robot sin ninguna molestia, estas capacidades son las mejores opciones de implementación, tal forma al escribir nuevas líneas de código es como volver inventar la rueda, todas estas capacidades son altamente configurables permitiendo afinar cada capacidad usando diferentes parámetros.
* **¿Toneladas de herramientas?**: Ros is a packed with various tools for debugging, visualization and performing simulation. The tool such as rqt\_gui, RViz and Gazebo are some of the strong open source tools for debugging, visualization and simulation.
* **Soporte de sensores de alta Gama y actuadores:** Ros esta lleno con driver de dispositivos y paquetes de interfaz de varios sensores y actuadores en robótica.
* **Operatividad Inter-Plataforma:** The ROS messaging-passing middleware allows communicating between different nodes. These nodes can be programmed in any language that has ROS client libraries. We can write high performance nodes in C++, C and other nodes in python or java. This kind of flexibility is not available in other frameworks.
* **Modularidad:** Uno de los temas (Problemas) que pueden ocurrir en la mayoría de las aplicaciones independientes del robot son, si uno de los hilos del código principal falla, toda la aplicación del robot puede ser parada. En ROS la situacion es diferente, nosotros escribiremos diferentes para cada uno de los procesos y si uno de los nodos falla, el sistema puede seguir trabajando.
* **Manejo de recursos concurrentes**

Manejo de recurso de hardware por mas de dos procesos es siempre un dolor de cabeza. Imaginemos, Nos. Deseamos procesar una imagen de una camara para deteccion de rostros y deteccion de movimientos, nosotros podemos ya sea escribir el codigo como una sola entidad que pueda hacer ambos, o Nos. Podemos escribir una simple cadena de codigo para concurrencia. Si Nos. Deseamos agregar mas de dos caracteristicas en las cadenas el comportamiento de la aplicación podria ser compleja y podria ser dificil para depurar. Pero en ROS, se puede acceder a los dispositivos usando ROS TOPIC desde los DRIVERs ROS. Cualquier numero de ROS NODES pueden suscribirse al mensaje de la imagen desde el Driver de camara ROS y cada nodo puede optimizar diferentes funcionalidades. Esto puede reducir la complejidad de los recursos usados para el proceso (Computation) y ademas incrementar la depuracion-capacidad(ability) de el sistema entero.

* **Comunidad activa**

**3. Install Ros kinetic**

El paquete ROS no esta soportado oficialmente en la librería de paquetes de UBUNTU para eso es necesario instalarlo desde un link externo.

***>>****sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -cs) main" > /etc/apt/sources.list.d/ros-latest.list'*

Se crea el directorio ros-latest.list para el soporte de los paquetes que conforman ROS ademas que ciertos repositorios necesitan de instalación de Claves de autorización o .key (APT necesita de una clave GPG para obtener de ciertos paquetes)

[Ejemplo de comandos >>*sudo apt-key add public-key-file* or *wget -qO - http://example.com/archive.key | sudo apt-key add -*]

>> *wget http://packages.ros.org/ros.key -O - | sudo apt-key add -*

>>*sudo apt-get update*

>>*sudo apt-get install ros-kinetic-desktop-full*

*>>sudo apt-get install ros-kinetic-ros-control ros-kinetic-ros-controllers*

*>>sudo apt-get install ros-kinetic-map-server ros-kinetic-fake-localization*

*>>sudo apt-get install ros-kinetic-pr2-common*

>>sudo rosdep init

>>rosdep update

>>source /opt/ros/kinetic/setup.bash

>>echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc //Existeb usuarios que poseen varias distribuciones de ROS con estos comandos permites inicializar con cual se trabaja.

ADD:) Español

source /opt/ros/kinetic/setup.bash

source ~/ros\_qtc\_plugin/devel/setup.sh

>>source ~/.bashrc

.basch

Directorio del sistema ROS

nano .bashrc

source ~/ros\_qtc\_plugin/devel/setup.sh

**4. Install QT Creator**

wget http://download.qt.io/official\_releases/qt/5.9/5.9.1/qt-opensource-linux-x64-5.9.1.run

chmod +x qt-opensource-linux-x64-5.9.1.run

./qt-opensource-linux-x64-5.9.1.run

wget <http://download.qt.io/official_releases/qt/5.7/5.7.0/qt-opensource-linux-x64-5.7.0.run>

**Install**

**Adjust permission, run the installer and follow the instruction to complete the installation.**

chmod +x qt-opensource-linux-x64-5.7.0.run

./qt-opensource-linux-x64-5.7.0.run

<https://github.com/ros-industrial/ros_qtc_plugin/wiki/2.-Improve-ROS-Qt-Creator-Plugin-(Developers-ONLY>)

*sudo add-apt-repository ppa:levi-armstrong/qt-libraries-xenial*

*sudo add-apt-repository ppa:levi-armstrong/ppa*

*sudo apt-get update && sudo apt-get install qt57creator-plugin-ros*

### Configure system to use the new version of Qt

### >> s*udo gedit /usr/lib/x86\_64-linux-gnu/qt-default/qtchooser/default.conf*

### Replace

### */usr/lib/x86\_64-linux-gnu/qt4/bin*

### */usr/lib/x86\_64-linux-gnu*

### For

### */opt/qt57/bin*

*/opt/qt57/lib*

Git clone

>>git clone -b master <https://github.com/lbellonda/qxmledit>

>>qmake

>>make

>>sudo make install

>> git clone -b master <https://github.com/ros-industrial/ros_qtc_plugin.git>

Una vez teniendo la rama del repositorio se puede actualizar manualmente ejecutando el comando

>>bash setup.sh -d

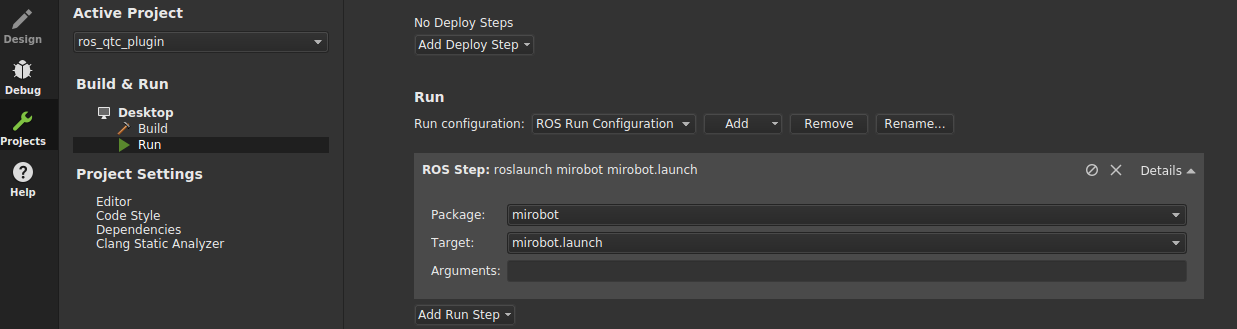
\*

Despues de ejecutar todas las instrucciones abrimos el archivo .workspace en /ros\_qtc\_plugin/ros\_qtc\_plugin.workspace con QT creator.

Creamos el espacio de trabajo en QT creator.

Tenemos dos opciones la de debugging y Play si creamos un nuevo repositorio entonces podemos correr este mismo utilizando RUN e insertando el espacio de trabajo.

Ejemplo.



### ***5. Teoria***

### *Un nodo con C++ .cpp*

### *-*

### *-*

### *-*

### *Que es un .XML*

### *-*

### *-*

### *-*

### *Que es un .Launch*

### Archivos .launch son muy coomunes en ros para usuarios y desarrolladores. Proporcionan una manera conveniente de arrancar múltiples nodos y un maestro, así como otros requisitos de inicialización como parámetros de configuración.

### RosLaunch es usado para abrir archivos .launch. Estos puede hacerce especificando el paquete en el que se encuentran los archivos de lanzamiento, seguido por el nombre del archivo de inicio o especificando la ruta del archivo del archivo de lanzamiento.

### Los archivos launch poseen el formato .launch y usan en su sintaxis un formato XML especifico. Pueden colocarse en cualquier lugar dentro de un directorio de paquetes, pero es común crear un directorio denominado "LAUNCH" dentro del directorio del espacio de trabajo para organizar todos los archivos de inicio

### Que es un .world

### -

### -

### -

### Difference between URDF and SDF

### URDF (Universal Robot Description Format) puede solo especificar las propiedades cinemática y dinámica de un simple robot en aislamiento.

### -URDF no puede especificar la posición del robot junto al mundo de trabajo.

### - No puede especificar objetos que no sean robots como luces, mapas etc.

### -Carece de definir propiedades de fricción.

### SDF (Simulator description Format) Es una completa descripcion para todo el nivel del MUNDO y el nivel del ROBOT

### *(Esta fase de decidir que formato SDF/URDF utilizar es importante en los detalles del marco teorico)*

The Filesystem level

Packages: Crean o organizan la mínima estructura y contenido para la creación de un programa junto con ROS (Podemos ver esto como un atomo Centro ROS y electrones los paquetes y herramientas que giran alrededor de el) Un paquete puede contener procesos de tiempo de ejecución de ROS (nodos), una biblioteca dependiente de ROS, conjuntos de datos, archivos de configuración o cualquier otra cosa que esté organizada de manera útil en conjunto

Al hacer un paquete se le puede agragar las dependencias directamente desde la linea de comandos. >> *catkin\_create\_pkg <package\_name> [depend1] [depend2] [depend3]*

Packages manifests: Proveen la información sobre un paquete, licencias, dependencias etc. Es administrado bajo el archivo llamado *package.xml*

*Estructura del folder.*

### *Para ver el estado de archivos, carpetas y variables de ROS*

### *>> printenv | grep ROS printenv (Print part or all of the environment)*

### *grep (global regular expression print)*

### *Cada vez que se haga un repositorio nuevo es necesario ejecutar el comando siguiente*

*>> source /opt/ros/kinetic/setup.bash*

### FileSystem Tools

### **Rospack**

### Es una herramienta de linea de comando para la recuperacion de informacion sobre paquetes de ROS disponible en los archivos del sistema.

**Navegacion sobre el entorno**

* **Encontrar un paquete**

>> *rospack find [package\_name]*

*Este arroja la locación de la carpeta*

* **Dirigirse al directorio del paquete**

>> r*oscd [locationname[/subdir]]*

Se dirige al directorio.

* **Ver archivos de paquete sin estar en el directorio**

>>*rosls [locationname[/subdir]]*

Muestra archivos sin necesidad de dirigirse al directorio

## Create a ROS Workspace

Creando el Workspace

*>>mkdir -p ~/catkin\_ws/src*

*>>cd ~/catkin\_ws/src*

*>>catkin\_init\_workspace*

Cosntruyendo el espacio de trabajo

>> cd ~/catkin\_ws/

>> catkin\_make

Como nota en esta sección al crear el WORKSPACE en el ambiente de trabajo de ros Se actualiza que el directorio de trabajo de ROS esta en (/*home/udp/catkin\_ws/*)

Creating a ROS package

Dirigiendose al entorno de Trabajo (*/catkin\_ws/src*)

Creando el primer paquete de trabajo a este es necesario describirles las dependencias que se utilizaran.

>> *catkin\_create\_pkg <package\_name> [depend1] [depend2] [depend3]*

Es necesario construir el paquete (compilar) que se encuentran dentro del workspace

*>>cd ~/catkin\_ws*

*>>catkin\_make*

Para agregar el espacio de trabajo al entorno ROS, debe generar el archivo de instalación generado

>>*. ~/catkin\_ws/devel/setup.bash*

*Ejecutando Nodos ROS*

Para iniciar ROS

>> *roscore*

Ejecutando la herramienta TurtleSim

>> *rosrun turtlesim turtlesim\_node*

Vincularemos el teclado de la PC de este modo permite mostrar la operabilidad del entorno ROS con el Hardware para esto usaremos el nodo.

*>> rosrun turtlesim turtle\_teleop\_key*

Ver los diferentes nodos

*>> rosnode list*

Si se desea ver el estado de los diferentes topics que contienen mensajes entonces ejecutamos

>> *rostopic list*

*Para ver informacion de Nodos.*

>> *rostopic echo /turtle1/cmd\_vel*

Se puede observar el tipo de mensaje que envia el topic usando el comando

*>> rostopic type /turtle1/cmd\_vel*

>>*Geometry\_msgs/Twist Se obtiene esta información*

Para ver el contenido del mensaje entonces

>> *rosmsg show geometry\_msgs/Twist*

*>> rostopic pub -1 /turtle1/cmd\_vel geometry\_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'*

Esto altera el mensage que recibe TurtleSim (Con …/Twist - r 1 - - ‘[2.0 3.0…) se mantiene constantemente alterando el mensage.

Para Observar la informacion de los Topics proveniente de los nodos Tecleamos

*>> rosrun rqt\_graph rqt\_graph*

Si deseamos ver el estado de los mesajes de los Topics

*>> rosrun rqt\_plot rqt\_plot*

Tuto1

Inicializando ROS

>>*source /opt/ros/kinetic/setup.bash*

*Creando el directorio*

*>>mkdir ~/robot > cd robot*

*>>mkdir src > cd src*

*>> catkin\_init\_workspace >cd ..*

*>>catkin\_make*

*>>**source ~/robot/devel/setup.bash*

*rosrun test2 listener*

INICIANDO CUALQUIER TUTO

Ir a carpeta donde esta alojada la librería de ROS

**Tuto2** *Creación de paquetes pueden ir muchos*

>>source *~/ros\_qtc\_plugin/devel/setup.bash*

*>>catkin\_create\_pkg -D “Descrpicion del paquete” -a Yesser --rosdistro KINETIC mirobot*

*Carpeta Launch*

*Entendiendo Launch Ir a teoria .launch* editar y poner las lineas ARG o INCLUDE de los distintos programas a utilizar –Gazebo

*<include file="$(find gazebo\_ros)/launch/empty\_world.launch">*

*<arg name="world\_name" value="$(find mirobot)/worlds/mirobot.world"/>*

*>>roslaunch mirobot mirobot.launch*

*Edicion del archivo XML / SDF .world para creacion del modelo en gazebo*

*Tuto3*

Plugin Gazebo and SDF

Es necesario acceder a la referencia de los plugin’s de gazebo:

http://osrf-distributions.s3.amazonaws.com/gazebo/api/7.1.0/classgazebo\_1\_1ModelPlugin.html

Incluir librería: #include<xxxxx.hh>

-Se edita el archivo package.xml para integrarles las dependencias necesarias para el diseño si no se agregaron desde la creación del paquete de trabajo.

### *catkin\_create\_pkg <package\_name> [depend1] [depend2] [depend3]* o

<build\_depend>roscpp</build\_depend>

<build\_depend>gazebo\_ros</build\_depend>

en el archivo .xml-Se edita el archivo Cmake agregando las instrucciones necesarias para que el compilador encuentre los paquetes para luego ejecutar la compilación.

find\_package(gazebo REQUIRED)

-Se agrega las instrucciones sobre la lista de dependencias de los paquetes con el compilador.

(

DEPENDS

roscpp

)

-Luego se especifica el directorio de las librerias para que el compilador las encuentre.

link\_directories(${GAZEBO\_LIBRARY\_DIRS})

### include\_directories(

### include

### ${Boost\_INCLUDE\_DIR}

### ${catkin\_INCLUDE\_DIRS}

### ${GAZEBO\_INCLUDE\_DIRS}

### )

### -Se declaran las libreras a usar en C++ en el proyecto indicando la ubicación

### add\_library(${PROJECT\_NAME}

### codigo/miRobot.cpp

### )

### -Establecemos los enlaces simbolicos entre las librerias del proyecto y el compilador

### target\_link\_libraries(${PROJECT\_NAME}

### ${catkin\_LIBRARIES}

### )

### Despues hay una linea un tanto interesante define el tipo de C++ a utilizarse en este caso es

### add\_definitions(-std=c++11)

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### .SO es un archivo Shared Object similar a DLL format de Windows.

### Esta librería .SO se crea por la compilacion del C++ creado una vez indicado las funcionalidades a Catkin\_make de las **dependencias** a utilizar en el project

### Despues se vincula con el MODEL de gazebo (.world) agregando

<plugin filename="libMyPlugin.so" name="my\_plugin"/>

Se crea una clase que enlace el Cpp con gazebo ingresando el siguiente comando para registrarlo.

//GZ\_REGISTER\_MODEL\_PLUGIN(Mirobot)

Seguimos editando el .Cpp y en la Clase Creada “Mirobot” agregamos

Funciones virtuales para cargar Macros etc. asociados con el Model y el SDF

*Tuto4*

Talker and listener node With Gazebo.

Ros se comunica ya sea de manera fisica o virtual con el robot.

*#ifndef LISTENER*

*#define LISTENER*

*#endif*

*Permite declarar solo una vez la clase. Una sola carga!*

PackML GUI RQT

Cargando robot industrial

En el workspace de trabajo de ros\_qtc\_plugin

>> git clone <https://github.com/ros-industrial/universal_robot.git>

>>accediendo al workspace SRC

***RVIZ with Qtcreator***

*FUENTE:*

*http://docs.ros.org/kinetic/api/librviz\_tutorial/html/index.html*

*Rviz no es solo una aplicación para visualización es ademas una librería. Puede ser accedida en una aplicación propia o una GUI*

*Usando un Qt slider control se accedera al ajuste de las propiedades de dos grid’s*

***RVIZ with MoveIt***

Install Ubuntu Packages

Simply run:

>> sudo apt-get install ros-kinetic-moveit

>> source /opt/ros/kinetic/setup.bash

>> rosdep update

>> sudo apt-get update

>> sudo apt-get dist-upgrade // Upgrade new package and delete later versions.

**Working with Plugin**  *(Mastering ROS for Robotic Programming)*

*In ROS is possible work with Pluginlib, Nodelets and Gazebo Plugins.*

**Understanding Pluginlib**

The advantage of plugins are that we don’t need to write all the features in main software, instead, we can make an infrastructure on the main software to accept new plugins to it. Using this method, we can extend the capabilities of software to any level.

We need plugins for our robotics application too. When we are going to build a

complex ROS based application for a robot, plugins will be a good choice to extend

the capabilities of the application.

The ROS system provides a plugin framework called pluginlib to dynamically load/

unload plugins. Plugin files are runtime libraries such as shared objects (.so) or dynamic link

libraries (.DLL).

The main advantage of plugins is that we can expand the application capabilities

without making many changes in the main application code.

Follow and example control PR2 Gazebo

1. Making package

>>catkin\_create\_pkg my\_controller\_pkg roscpp pluginlib pr2\_controller\_interface pr2\_mechanism\_model

2. Add my\_controller\_file.h

**Working with rosparam**

**Obtaining info of Subscriber**